UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19|20

Trabajo fin de grado

Interacción con usuario para reporte y diagnóstico de incidencias

Judith Manchón Vállegas

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

excelencia Campus Internacional UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Doble Grado en Ingeniería Informática y Matemáticas

TRABAJO FIN DE GRADO

Interacción con usuario para reporte y diagnóstico de incidencias

Si hace falta subtítulo

Autor: Judith Manchón Vállegas
Tutor: Daniel Perdices Burrero
Ponente: Javier Aracil Rico

junio 2020

Judith Manchón Vállegas
Interacción con usuario para reporte y diagnóstico de incidencias

Judith Manchón Vállegas
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mi hermano

Cada día sabemos más y entendemos menos.

Albert Einstein

Agradecimientos

En primer lugar, agradezco a naudit HPCN la oportunidad que me ha brindado con el desarrollo de este trabajo y todo lo que he aprendido gracias a los miembros del grupo de investigación HPCN. Le agradezco sobretodo a Daniel Perdices haber sido un gran tutor que me ha guiado durante todo el proyecto y me ha prestado ayuda en todo momento.

Por otro lado, quiero agradecer a mi familia el amor y apoyo que me han dado durante todos estos años, así como la paciencia durante los meses de estrés, en especial a mi madre, que siempre me ha cuidado cuando lo he necesitado. Gracias a Merche por su cariño y apoyo.

Agradezco también a Patricia toda la ayuda recibida, lo que me ha enseñado y las horas que hemos pasado hablando.

Por último, le agradezco a mis amigos estar ahí siempre que los necesité, que me animasen los días malos y mejorasen los buenos. En especial, a Emilio y las infinitas tardes de estudio en la biblioteca, a Roberto por intentar enseñarme inglés, a Elena por ser tan buena amiga y a Lucas por estar siempre.

Resumen

En los últimos años, las empresas han introducido nuevos modelos de negocio, incorporando más equipos informáticos y nuevas tecnologías, evolucionando así a la llamada industria 4.0. Por tanto, la asistencia técnica para resolver problemas informáticos supone no solo una actividad crítica sino un elevado gasto en personal o contratación externa. En este contexto, los sistemas de recogida y gestión de incidencias permiten optimizar el ciclo de vida de la incidencia, esto es, el proceso desde que se genera la incidencia hasta que es recogida y resuelta por el analista. En este trabajo, se presenta un nuevo modelo de gestión de incidencias que permite la recopilación de los datos sin necesidad de personal dedicado a ello, consiguiendo una gran usabilidad por parte del usuario que la reporta. Presenta también la ventaja de poder acceder a ella incluso cuando la red local de la empresa no funcione, de modo que cualquier usuario con teléfono móvil puede reportar una incidencia desde éste. El sistema que se presenta en este trabajo no solo recopila y procesa los datos si no que, mediante el algoritmo tf-idf, el analista recibe también una lista de incidencias similares resueltas con anterioridad y la forma en que se solucionaron.

Se propone una arquitectura que enlaza un canal de entrada para la recogida de incidencias con un gestor de chat, que procesa los datos para facilitar la información al analista, encargado de resolver las incidencias. Asimismo, cuenta con un sistema experto que aporta cierta inteligencia al proyecto.

Para evaluar la aplicación, se ha expuesto al sistema a conversaciones basadas en problemas comunes a las que se enfrentan usuarios de empresas. La aplicación ha demostrado mantener la lógica de la conversación, almacenamiento correcto de datos y recomendación correcta de incidencias similares. Además, se ha sometido al sistema a conversaciones simultáneas y se ha comprobado su coherencia y efectividad.

En la fase de pruebas con usuarios, hemos demostrado una mejora de la usabilidad, puesto que encontraban más sencillo e intuitivo nuestro sistema respecto a otros que habían utilizado. Desde el punto de vista del analista, hemos recibido comentarios positivos dado que el sistema recopilaba correctamente los datos esperados y realizaba recomendaciones adecuadas. Una vez se ha implementado y probado la efectividad de esta prueba de concepto, que era el objetivo de este trabajo, hemos logrado sentar las bases para poder aplicar este modelo de gestión en un entorno real.

Palabras clave

Gestión de incidencias, usabilidad, procesamiento del lenguaje.

Abstract

During the last years, companies have introduced new business models, incorporating new computer equipment and new technologies, leading to an evolution to the so called 4.0 industry.

Therefore, technical assistance to resolve computer problems is not only a key activity but also a high expense in personal or external resources. In this context, the managing and gathering systems of incidents allow to optimize the life cycle of the incident, this is, the process from generating the incident until its attended and resolved by the analyst. In this work, we present a new incident management model which allow us to gather data without the need of dedicated personal, achieving a great usability from the reporting user. It also presents an advantage of being able to access the system even when the local net of the enterprise is not working, therefore any user with a smartphone can report an incident from his phone. The system presented in this work not only collects and process the data, but though the tf-idf algorithm, the analyst receives a list of similar incidents which had been resolved and the process followed to solve it.

This work proposes an architecture which links the input channel for the incidents gathering and the manager of the chat, which process the data for the analyst. Furthermore, it includes an expert system which provides some intelligence to the project.

In order to evaluate the application, the system had been exposed to conversations of in everyday problems. The application has proven to maintain the logic of the conversation, storage of the data and provide a correct recommendation of similar incidents. Furthermore, it had been tested with simultaneous conversations proving to be coherent and effective.

In the testing phase with users, we've proven that it improves its usability, given that they found our system easier to use and more intuitive than other used systems. From the analyst point of view, we've received positive comments referring the correct storage and correct recommendations. Once it was implemented and proven its effectivity of this concept test, which was the goal of this work, we've achieved to give the ground truths to apply this managing model in a real environment.

Keywords

Incident management, usability, language processing.

Índice

Listas

Lista de códigos

Lista de ecuaciones

Lista de figuras

Lista de tablas

1

Introducción

1.1. Motivación

En las últimas décadas, las compañías han experimentado grandes cambios a nivel del uso de recursos informáticos y nuevas tecnologías. En 2019, el 79.91 % de las empresas de menos de diez empleados utilizan medios informáticos para el desarrollo de su negocio y el 76.31 % tenían parte de su negocio en Internet. Esta tendencia es mucho más acentuada en el caso de las grandes empresas, el 99.26 % usan sistemas informáticos a diario y el 98.39 % Internet. En muchos casos, el acceso a la red es parte fundamental de su desarrollo comercial, realizando compras o ventas a través de comercio electrónico (20.36 % y 33.90 % respectivamente) que suponen además una importante parte de su facturación [1].

Este desarrollo tecnológico ha generado un impacto económico positivo y ha permitido la creación de nuevos nichos de mercado, dando lugar tanto a la creación de pequeñas y medianas empresas (PYMES) y grandes empresas en el sector, dirigidas a responder a las nuevas necesidades, como a la modernización de empresas que ya conformaban el tejido empresarial. En 2017 el sector de Tecnologías de la Información y la Comunicación (TIC) experimentó un aumento del 7.4 % respecto al año anterior, alcanzando un volumen de negocios de 96750,3 millones de euros [1].

Las Proveedoras de Servicio de Internet (*Internet Service Provide*, ISP) así como las empresas que provisionan su infraestructura y servicios de telecomunicaciones requieren de continua asistencia técnica para prestar el servicio a sus clientes y mantener activo su negocio. La asistencia técnica se lleva a cabo a través de personal contratado por la empresa o por empresas externas del sector tecnológico que dan servicio de asistencia. En el primer trimestre de 2019, el 17.4 % de empresas emplearon a especialistas TIC para llevar a cabo tareas de mantenimiento o asistencia de sus infraestructuras o servicios. Algunas de las mayores empresas que ofrecen asistencia interna (a sus empleados) o externa (a sus clientes), basan sus servicios en la recopilación de datos de incidencias en consultas telefónicas y/o sistemas de gestión de *tickets*. Estas metodologías llevan asociado un alto gasto directo en personal en la recopilación, tratamiento y gestión de datos.

Los sistemas de gestión automatizados, sin personal dedicado a la recolección de los datos, ofre-

cen una alternativa que permite la recopilación de las incidencias, su filtrado y procesado, facilitando el trabajo de análisis de la incidencia. A pesar de que ya existen soluciones enfocadas a este fin, la poca usabilidad de las aplicaciones actuales ha dificultado la implantación masiva en las empresas.

Este proyecto presenta un nuevo modelo de gestión de incidencias que permite la recopilación automatizada de datos, la respuesta basada en experiencias anteriores y acceso mediante aplicación móvil y página web. De esta forma, se pretende mejorar la experiencia del usuario y agilizar la respuesta del analista o gestor, facilitando así, la implantación de la tecnología por parte del tejido empresarial.

1.2. Objetivos

El objetivo principal de este proyecto es el diseño y desarrollo de un sistema autónomo que recoja y procese datos de incidencias de los usuarios, sin necesidad de personal dedicado a esta labor. El sistema debe gestionar todo el ciclo de vida de las incidencias, incluyendo dirigirlas a los analistas.

Además, este proyecto se centrará en la usabilidad del sistema. La aplicación está diseñada para que sea más cómoda, rápida y fácil de usar que los sistemas tradicionales. Para ello, no solo empleará lenguaje natural, si no que la aplicación podrá utilizarse a través de distintas plataformas, como dispositivos móviles o la propia web corporativa.

La incidencia recogida se comparará con incidencias anteriores similares para recomendar distintas soluciones que respondan a los problemas del usuario en base al conocimiento previo obtenido en incidencias pasadas, para así proveer de una posible solución cuanto antes.

Por último, el envío de alarmas e incidencias a los analistas se llevará a cabo a través de plataformas comunes, como el correo o los chats corporativos (e.g. Microsoft Teams, Slack). El sistema, además, debe recoger las respuestas y soluciones a estas incidencias que provea el analista.

1.3. Plan

El trabajo a realizar se ha dividido en los siguientes hitos principales:

- Análisis
- Diseño
- Desarrollo
- Pruebas, correcciones y mejoras
- Presentación final

A continuación, en la tabla 1.1 podemos ver cada una de las tareas asignadas a los hitos, así como su duración estimada junto con las fechas de inicio y fin previstas para cada una de estas tareas.

Nombre de la tarea	Duración	Comienzo	Fin
Análisis			
Reuniones iniciales y recopilación de los requisitos	4h	14/10/2019	15/10/2019
Análisis de los requisitos	4h	16/10/2019	17/10/2019
Diseño de arquitectura previa	2h	18/10/2019	18/10/2019
Tiempo total del hito	**10**		
Diseño			
Búsqueda de información y elección de tecnologías	32h	21/10/2019	12/11/2019
Diseño de la arquitectura	2h	13/11/2019	13/11/2019
Diseño de las conversaciones	10h	14/11/2019	20/11/2019
Diseño de clases	16h	21/11/2019	02/12/2019
Diseño de la estructura de los datos de la BD	6h	03/12/2019	05/12/2019
Reunión de finalización de fase	1h	10/12/2019	10/12/2019
Tiempo total del hito	**67**		
Desarrollo			
Aprendizaje de nuevas tecnologías y lenguajes	30h	10/12/2019	02/01/2020
Implementación del gestor de chat	45h	02/01/2020	04/02/2020
Desarrollo de Bigía	24h	05/02/2020	20/02/2029
Instalación, configuración y llenado de la BD	10h	21/02/2020	27/02/2020
Configuración de Botpress	8h	28/02/2020	04/03/2020
Configuración de los canales de entrada	4h	05/03/2020	06/03/2020
Conexión del gestor de chat con el canal de salida	2h	09/03/2020	09/03/2020
Despliegue de la aplicación mediante contenedores	8h	10/03/2020	13/03/2020
Reunión de finalización de fase	1h	16/03/2020	16/03/2020
Tiempo total del hito	**132**		
Pruebas, correcciones y mejoras			
Pruebas, correcciones y mejoras	40h	16/03/2020	15/04/2020
Reunión de finalización de fase	1h	15/04/2020	15/04/2020
Tiempo total del hito	**41**		
Presentación final			
Escritura de la memoria	40h	16/04/2020	14/05/2020
Realización del material de la presentación	8h	18/05/2020	21/05/2020
Reunión final	1h	14/05/2020	14/05/2020
Presentación ante la comisión evaluadora	1h	X/06/2020	X/06/2020
Tiempo total del hito	**50**		
Tiempo total	**300**		

Tabla 1.1: Planificación del trabajo mediante hitos y tareas.

Esta previsión se ha realizado teniendo en cuenta que se dedicarán dos horas diarias al desarrollo del trabajo. Sin embargo, gracias a la flexibilidad permitida por la empresa, las fechas de las tareas fueron orientativas permitiendo dedicar el tiempo requerido los días que mejor se ajustaban a mi situación académica. Las fechas de los hitos, por otro lado, fueron bastante más estrictas para asegurar la finalización del trabajo en la fecha indicada.

Es importante destacar que el tiempo de aprendizaje no se distribuyó como se esperaba en un principio. El tiempo que dediqué a conocer el lenguaje nodejs no fue tan largo como imaginé, es decir, al menos la mitad del tiempo esperado para la fase de Aprendizaje de nuevas tecnologías y lenguajes(30h). Sin embargo, el despliegue de la aplicación mediante contenedores fue más compleja de lo esperado debido a que era una tecnología completamente nueva para mi.

Podemos ver este desarrollo de forma más visual mediante el diagrama de Gantt representado en las figuras 1.1 y 1.2.

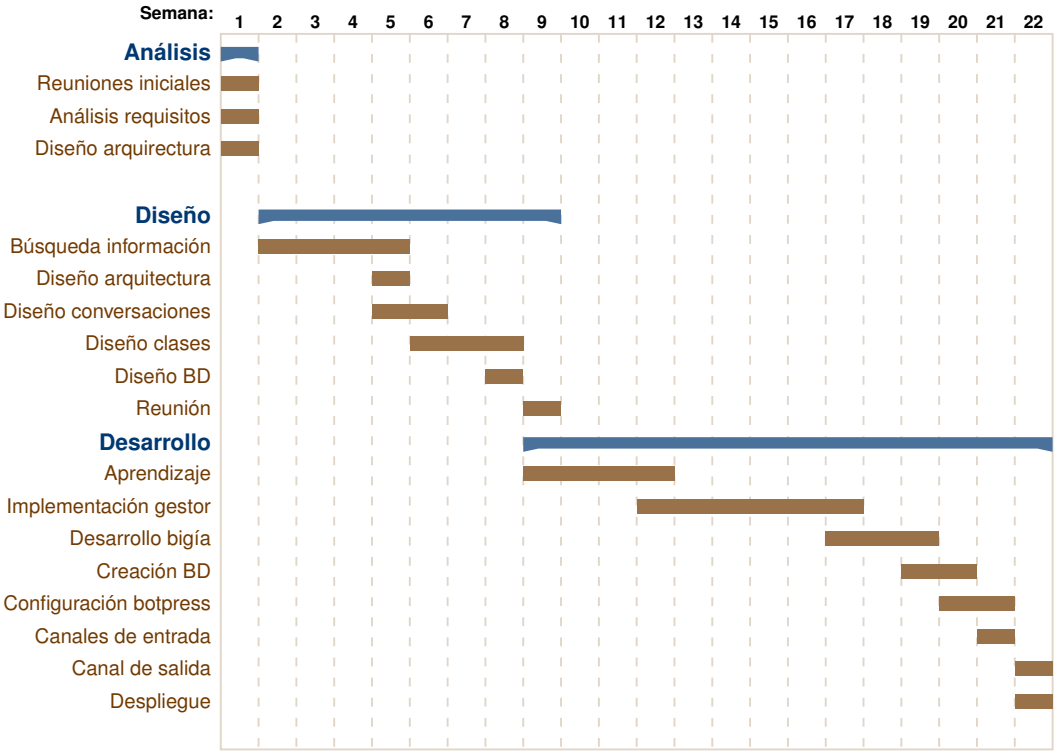

Figura 1.1: Diagrama de Gantt donde se muestra el tiempo previsto para los hitos del trabajo y sus respectivas tareas (primera parte)

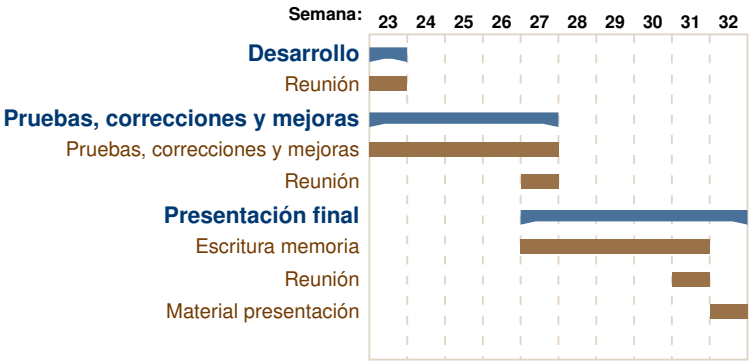

Figura 1.2: Diagrama de Gantt donde se muestra el tiempo previsto para los hitos del trabajo y sus respectivas tareas (segunda parte)

Este proyecto se va a realizar a través de un desarrollo iterativo e incremental, es decir, va a contar de distintas etapas repetitivas, las cuales, cada una de ellas, finalizarán con una versión funcional. En este trabajo se va a desarrollar la primera iteración o etapa, pero está preparado que pueda seguir desarrollándose, mejorando y añadiendo requisitos y funcionalidades al sistema.

Cada etapa será lineal, es decir, se desarrollará completamente y se probará al finalizar, exceptuando las pequeñas mejoras que se añadan tras las pruebas con usuarios.

1.4. Estructura de la memoria

Hemos comenzado presentando el proyecto, para ello también se han definido los objetivos a cumplir y la organización del trabajo que se ha llevado a cabo.

A continuación, mostramos algunos de los sistemas de gestión de incidencias actuales para poner en contexto al lector. Se explicarán además, algunas de las tecnologías empleadas para desarrollar nuestro servicio.

En el tercer capítulo se encuentra el análisis que se ha llevado a cabo, compuesto por los requisitos funcionales, los requisitos no funcionales y una primera idea de la arquitectura del proyecto.

Después, en el cuarto capítulo, pueden encontrarse algunas de las decisiones técnicas tomadas durante el diseño y el desarrollo del sistema, es decir, los lenguajes y tecnologías empleados, la arquitectura final del proyecto y la descripción del diseño interno de sus componentes.

Una vez se ha mostrado de forma detallada el sistema, mostramos las distintas pruebas realizadas y sus resultados.

Para finalizar, en el último capitulo se encuentran las conclusiones que hemos obtenido tras diseñar,

desarrollar y probar el sistema.

2

Estado del arte y tecnologías empleadas

2.1. Estado del arte

En la actualidad, se pueden encontrar bastantes tipos de sistemas encargados de recopilar los datos de las incidencias. Del mismo modo, existen múltiples sistemas que permiten gestionar las incidencias que puedan producirse. Como mostraré más adelante, estos sistemas no siempre realizan ambos procesos, si no que unos dependen de los otros.

Comenzaremos viendo aquellos sistemas cuya función es obtener los datos necesarios para que pueda resolverse la incidencia. Los hemos agrupado en cuatro tipos: automáticos, de correo electrónico, con operadores y de *tickets*.

Los sistemas automáticos son aquellos capaces de detectar un problema de forma autónoma y recopilar la información del sistema automáticamente. Esto permite que los datos de las incidencias se recojan prácticamente en tiempo real. Sin embargo, estos sistemas no son capaces de detectar todos los tipos de incidencias, si no que están pensados para problemas específicos. Un ejemplo de este tipo de sistemas sería Amazon CloudWatch [2] el cual está preparado para ser utilizado por usuarios con altos conocimientos en tecnología de la información (TI). Por otro lado este servicio permite definir parámetros para generar avisos y realizar ciertas acciones de forma automatizada. Podemos ver su funcionamiento en la figura 2.1. Este tipo de sistemas reduce los gastos en personal dedicado a esta tarea.

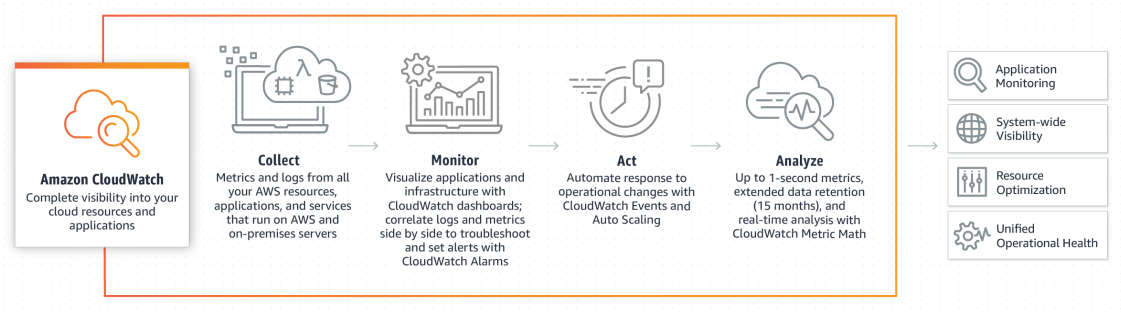

Figura 2.1: Funcionamiento del servicio Amazon CouldWatch.

Actualmente, existen multitud de grandes empresas que emplean el uso del correo electrónico para que el cliente notifique las incidencias. Mediante este método de reporte, al igual que el anterior, la empresa reduce gastos en personal, debido a que es el usuario el que se encarga de explicar el problema y recopilar los datos que considere necesarios de forma autónoma. El correo electrónico tiene la ventaja de ser conocido por una gran cantidad de usuarios, de modo que en principio, cualquier persona que sufra una incidencia podría reportarla. No obstante, estos usuarios pueden no tener conocimientos suficientes para adjuntar en dicho correo la información necesaria que se precisa para solucionarla. Esto puede deberse, en cierta medida, a que no existen campos de texto específicos que guíen al usuario.

Normalmente, muchas de las empresas que tienen opción de reportar las incidencias mediante correo electrónico, también ponen a disposición del usuario un número de teléfono. Esto permite que el cliente reporte su problema con ayuda de un operador que guíe la conversación con el objetivo de obtener los datos necesarios para solucionar la incidencia. Este operador podría estar formado llegando incluso a ser capaz de resolver el problema en el mismo momento en que se reporta. Sin embargo, este sistema produce un alto gasto para la empresa en personal que en muchas ocasiones no está siquiera formado para dar una solución, sino que únicamente recoge los datos y los envía al departamento correspondiente.

Un sistema que solucionaría algunos de los problemas que se producirían con el método del correo electrónico es el uso de *tickets*. Los *tickets* están formados por múltiples campos que permiten incluir la información de forma ordenada y completa. Además, estos *tickets* suelen contar con un ciclo de vida que informa del estado en que se encuentra el *ticket* en el momento en que se consulta, lo que puede resultar muy útil para aquel que esté intentando resolver la incidencia. Sin embargo, el usuario puede no poseer suficiente formación o conocimientos, de modo que no podría rellenar los campos solicitados por un *ticket* por sí mismo. La empresa que reporte las incidencias puede optar por contratar a un empleado con la suficiente formación para crear los *tickets* de las incidencias que puedan producirse, lo cual implicaría un gasto adicional. Un ejemplo de este sistema es Jira Service Desk [3] de Atlassian, en la figura 2.2 podemos ver su interfaz y un ejemplo de ciclo de vida de un *ticket*.

Por último, es importante recalcar que estos sistemas pueden estar combinados, por ejemplo, una vez que el operador ha recopilado los datos necesarios podría crear un *ticket* o mandar un correo electrónico a otro empleado con mayor formación.

Después de realizar la recopilación de los datos de la incidencia, se debe proceder a la gestión de ésta. Para seguir el ciclo de vida de estas incidencias, multitud de empresas han desarrollado diversos servicios que realizan esta labor. A continuación, mostraremos algunos de los servicios que nos han resultado interesantes.

Comenzaremos mostrando un servicio que utiliza de los sistemas automáticos mencionados anteriormente, entre otras herramientas de monitoreo. BigPanda [4] provee de un servicio que recoge los

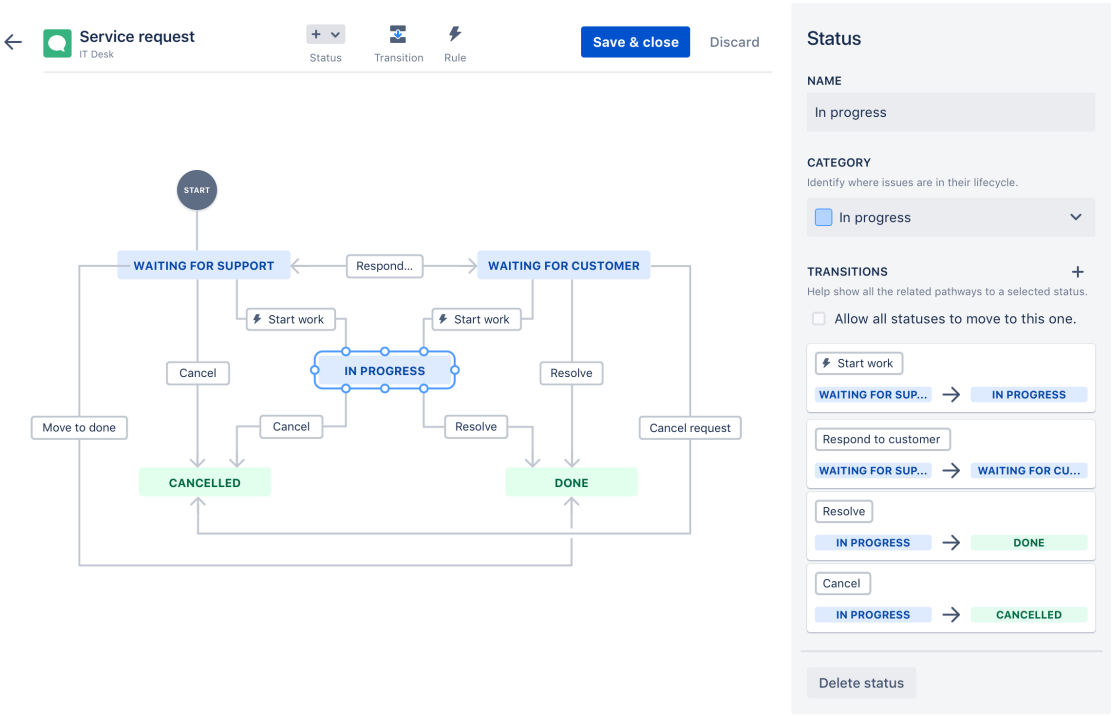

Figura 2.2: Ejemplo de creación y gestión de un *ticket* mediante el servicio Jira Service Desk de Atlassian

avisos o alertas que producen las herramientas de monitoreo y los filtra, eliminando un 95 % del ruido de TI, mediante un módulo propio de aprendizaje automático. Esto permite reducir el gasto en personal, como puede observarse en la figura 2.3, dado que se reducen el número de incidencias y falsos positivos. Igualmente, facilita el trabajo de los analistas y gestores debido a que identifica y muestra la causa más probable de la incidencia.

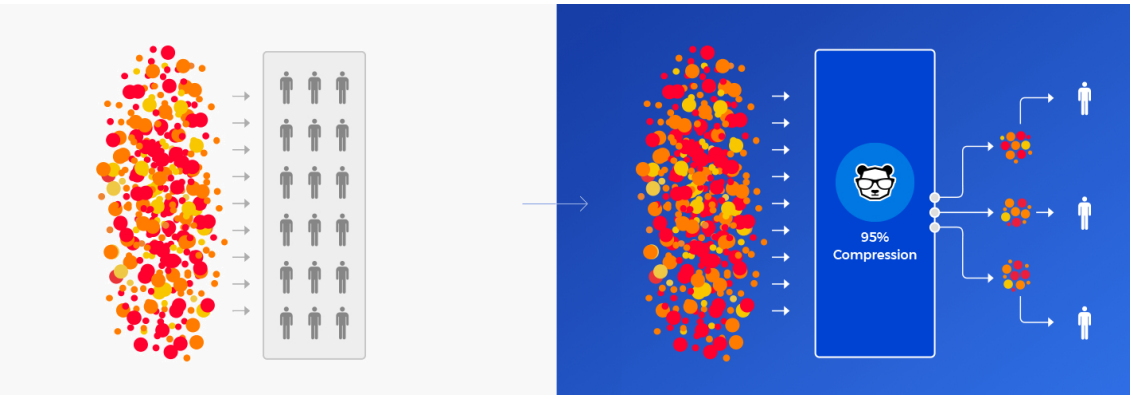

Figura 2.3: Esta imagen ilustra cómo el servicio BigPanda reduce el personal dedicado a resolver incidencias

Por otro lado, podemos encontrar Opsgenie [5] de Atlassian un gestor que permite crear filtros para eliminar ruido, agrupar las alertas y notificar a los analistas a través de distintos canales. Con este sistema se pueden fijar horarios en los que cada analista estará disponible y elegir las reglas para el

envío de las alertas en función de su fuente o importancia. Además analiza estas alertas y permite crear informes. Podemos ver en la figura 2.4 una parte de la interfaz que Opsgenie presenta para realizar las acciones mencionadas previamente.

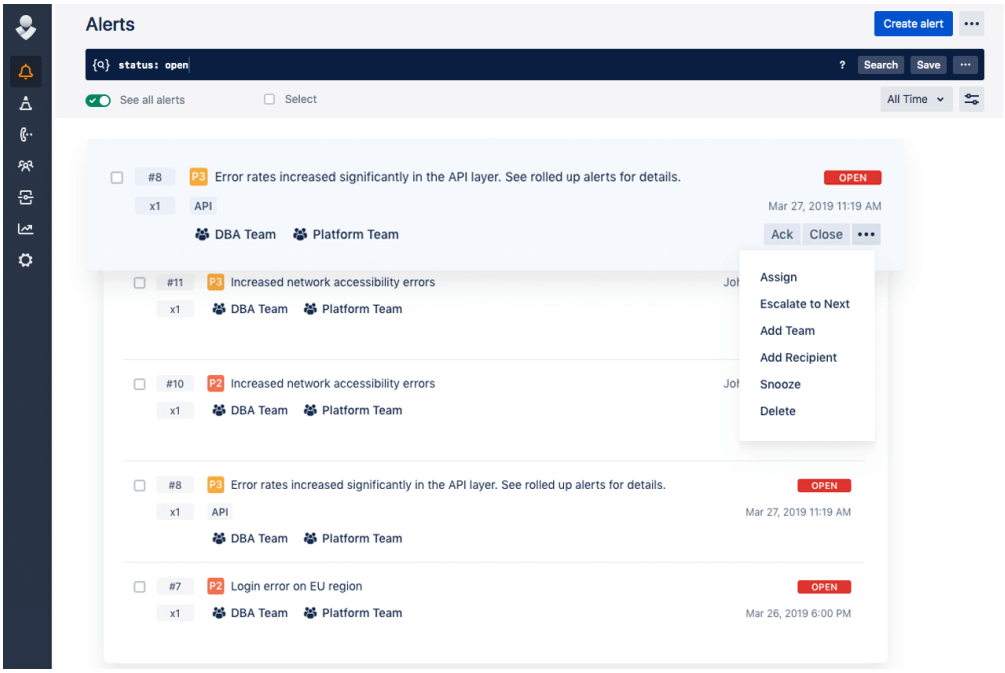

Figura 2.4: Interfaz del servicio Opsgenie para buscar y visualizar las alertas.

2.2. Tecnologías empleadas

En esta sección, vamos a mostrar las distintas tecnologías, librerías y lenguajes que se han valorado e investigado durante el trabajo. No se han empleado todas estas tecnologías debido a decisiones tomadas durante el diseño y desarrollo, pero de las no empleadas nombraremos solamente las que nos resulten más interesantes.

Empezamos con Node.js, un entorno de código abierto expresamente pensado para trabajar del lado del servidor. Este lenguaje está basado en JavaScript, construido con el motor JavaScript V8 de Chrome. Node.js es un entorno de ejecución orientado a eventos asíncronos y diseñado para desarrollar aplicaciones escalables. Existen muchas plataformas conocidas que utilizan este entorno, lo cual, de algún modo, determina si es una buena opción para este tipo de plataformas, por ejemplo Uber, Netflix y Wikipedia entre otras [6]. Además, Node.js es un entorno modular, por lo que normalmente se instalan distintos módulos o librerías, para lo cual es recomendable el uso el gestor de dependencias y software npm (*Node Package Manager*).

Existen multitud de módulos Node.js que pueden instalarse, desde simples funciones hasta complejas funcionalidades. Destacamos, entre estas librerías o módulos, Express, diseñado para crear

aplicaciones web y API REST. Express da soporte a distintos métodos de direccionamiento de HTTP, entre ellos *get* y *post*, lo que permite hacer API REST entre componentes de un proyecto. Con este módulo podemos direccionar distintas solicitudes y manejar las rutas de modo que, dependiendo de dónde venga la solicitud se realizarán unas acciones u otras.

Asimismo, Node.js puede ser combinado con bases de datos documentales que utilizan documentos para almacenar los datos. CouchBD [7] y MongoDB [8] son dos bases de datos documentales y de código abierto que emplean JSON(*JavaScript Object Notation* y BSON (JSON Binarios), respectivamente, para representar dichos documentos. Estas bases de datos son no relacionales, por lo que no utilizan lenguaje SQL para realizar consultas ni necesitan de estructuras fijas para almacenar sus datos.

Por otro lado, podemos encontrar servicios que automatizan el despliegue de aplicaciones. Este es el caso de Docker, un proyecto de código abierto que puede usarse para crear y gestionar contenedores, similares a máquinas virtuales pero que no requieren de un sistema operativo independiente como puede verse en la figura 2.5. El uso de estos contenedores permite aislar los distintos componentes, recursos y servicios. A través de docker-compose, podemos definir una aplicación mediante varios contenedores, es decir, nos permite desarrollar un proyecto con una arquitectura de microservicios.

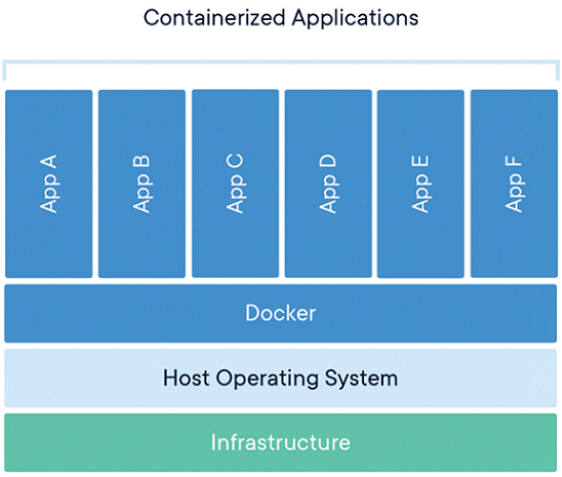

Figura 2.5: Contenedores Docker de distintas aplicaciones aisladas mediante el servicio Docker [9].

Para desarrollar *chatbots*, hemos empleado una plataforma de código abierto llamada Botpress. Botpress cuenta con una versión gratuita que permite crear *chatbots* mediante interfaces visuales para diseñar y desarrollar de forma sencilla y rápida flujos de conversación. También existe la opción de crear estos flujos a más bajo nivel con ayuda de las guías para desarrolladores y la documentación que ofrece. Además, no solo soporta múltiples plataformas de mensajería como Facebook y Telegram, si no que también permite conectarlo con un servicio web privado.

Para aportar cierta inteligencia al proyecto se han planteado dos opciones distintas y no necesaria-

mente excluyentes. Comenzaremos con Keras, una biblioteca escrita en Python de código abierto para redes de aprendizaje profundo [10]. Con esta biblioteca podríamos crear multitud de redes neuronales que permitirían reconocer una gran cantidad de patrones. Keras es utilizado por organizaciones tales como el Consejo Europeo para la Investigación Nuclear (CERN), la Administración Nacional de la Aeronáutica y del Espacio (NASA) y el Instituto Nacional de Salud (National Institutes of Healt, NIH) [11]. No obstante, podríamos utilizar un modelo vectorial con el esquema tf-idf. El esquema tf-idf [12] mide la importancia de un término para un documento dada una colección de documentos. Esta medida es el resultado de multiplicar la frecuencia de término (tf) y la frecuencia inversa del documento (idf), aunque hay multitud de posibilidades que permiten calcular estos valores. Con este modelo podemos conocer la correlación existente entre distintos documentos. Sin embargo, hay que tener en cuenta que este algoritmo solo contempla las frecuencias del texto, no su orden ni significado.

3

Análisis

Antes de comenzar un proyecto, es necesario comprender qué necesita el cliente y analizarlo de modo que podamos desarrollar más adelante la aplicación idónea que permita cubrir sus necesidades.

En este capítulo, se van a describir, mediante requisitos funcionales y no funcionales, las características que necesariamente debe cumplir la aplicación. Se definirán también los tipos de usuarios existentes y el rol que posee cada uno.

Por otro lado, tras analizar las funcionalidades, se plantea una primera arquitectura en la cual se muestran los componentes necesarios para llevar estos requisitos a cabo.

3.1. Requisitos funcionales

En esta sección, se muestran los distintos requisitos funcionales que debe abarcar nuestro proyecto. Para ello se detallan las distintas funcionalidades separándolas en cuatro grupos, aquellas que puede realizar cualquier usuario, las que están permitidas para clientes y analistas y, por último, las que el sistema debe suplir.

Antes de describir las funcionalidades, debemos destacar que existen tres tipos de usuarios:

- **Clientes:** la característica principal de los clientes es que poseen una dirección de correo corporativa que les permitirá, como veremos a continuación, identificarse. Estos usuarios serán los que podrán reportar sus incidencias.
- **Gestores y analistas:** del mismo modo que ocurre con los clientes, los gestores y analistas cuentan con una dirección de correo con la que, al identificarse, obtendrán el rol de “analistas”. El gestor o analista, a diferencia de los clientes, utilizará información aportada por el sistema para resolver incidencias. Una vez hecho esto, informarán a través del sistema del proceso elegido para solucionar el problema y otros datos de interés mostrados más adelante.
- **Otros usuarios:** existen otros usuarios que pueden hacer uso de este sistema, sin embargo, al no tener una dirección de correo que el sistema reconozca, no podrá identificarse. Es decir, estos usuarios podrán acceder, pero no obtendrán ningún beneficio del sistema.

3.1.1. Funciones generales

Los tres tipos de usuarios tendrán acceso a las funcionalidades mostradas a continuación:

RF-1.– Cualquier usuario podrá acceder al sistema mediante las distintas plataformas. Debe ser capaz de acceder a través de un dispositivo que al menos cuente con acceso a internet y, o bien, un navegador o una aplicación de mensajería móvil que sea compatible y esté conectada al sistema. Los ejemplos más comunes de estos dispositivos son un *smartphone*, un ordenador o una *tablet*. Para poder acceder mediante el navegador, el usuario tendrá que conocer la dirección de la página web que contenga la interfaz del chat.

RF-2.– Mediante el chat se permitirá que cualquier usuario envíe mensajes del sistema. Una vez el usuario ha accedido a la plataforma elegida podrá enviar un mensaje al sistema a través de la interfaz de chat.

RF-3.– A través de la interfaz del chat el usuario recibirá las respuestas del sistema. Después de que el usuario haya enviado un mensaje, la respuesta será mostrada en la misma interfaz de chat de modo que dicho usuario pueda leerlas.

RF-4.– Todo usuario podrá comenzar una conversación. Cualquier usuario tendrá la opción de comenzar una conversación escribiendo cualquier texto en la interfaz de chat y enviándolo.

RF-5.– Un usuario tendrá la posibilidad, si así lo desea, de terminar la conversación de forma abrupta. Si el usuario, por cualquier motivo, quiere terminar la conversación podrá hacerlo en cualquier momento enviando un mensaje con la palabra “terminar”. Si el usuario opta por terminar de este modo la conversación no quedará constancia en el sistema de dicha conversación.

RF-6.– Cualquier usuario tendrá la opción de solicitar ayuda cuando la necesite. Si el usuario envía un mensaje con la palabra “ayuda” el sistema le mostrará para que sirve el chat, lo que debe hacer para comenzar una conversación y cómo podría, si quisiera, terminar una conversación.

3.1.2. Funciones de los clientes

RF-1.– El cliente se identificará en el sistema con su dirección de correo corporativa. Una vez ha comenzado la conversación, el sistema solicitará la dirección de correo corporativa y, a continuación, el cliente deberá escribir su dirección de correo.

RF-2.– A través del chat, el cliente podrá reportar una incidencia. Si el cliente ya se ha identificado correctamente, el sistema comenzará a realizar preguntas que el usuario deberá responder si quiere informar de la incidencia. Con estas preguntas el cliente informará de la fecha y hora en que se produjo la incidencia, así como, el programa o servicio que estaba siendo utilizado cuando ocurrió.

RF-3.– Mediante la interfaz del chat, el usuario detallará el problema. Si el usuario ya ha informado de la fecha, hora y programa o servicio que estaba utilizando, deberá responder preguntas más concretas sobre la incidencia.

3.1.3. Funciones de gestores y analistas

RF-1.– El analista o gestor recibirá los datos necesarios de las incidencias que deba solucionar. El sistema enviará un mensaje a su dirección de correo corporativa. De este modo, los analistas pueden consultar los datos mediante su gestor de correo habitual. Además, en este mensaje se le proporcionarán soluciones de incidencias relacionadas de modo que le ayude a resolver el problema. Por otro lado, estos mensajes incluirán un identificador para cada incidencia.

RF-2.– El analista o gestor debe identificarse a través del chat. Una vez el gestor o analista ha iniciado una conversación mediante la interfaz del chat, el sistema solicitará la dirección de correo corporativa y, a continuación, el gestor o analista tendrá que escribir su dirección de correo.

RF-3.– El analista o gestor podrán informar al sistema tras resolver la incidencia. Una vez el gestor o analista ha considerado resuelta la incidencia tiene la opción de informar mediante el chat. Para ello, y después de haberse identificado en el sistema, tendrá que escribir el identificador de la incidencia resuelta. A continuación, el sistema realizará distintas preguntas a través de las cuales el analista o gestor debe mostrar si la incidencia ha sido resuelta, cómo ha procedido, las dificultades encontradas durante el proceso de resolución y el tiempo aproximado dedicado a esta labor.

3.1.4. Funciones que debe realizar el sistema

RF-1.– El sistema deberá tener almacenadas previamente las direcciones de correo electrónico de todos los clientes, gestores y analistas. Además de estas direcciones, contendrá otros datos relevantes como el rol de los usuarios a los que pertenecen dichas direcciones y, en caso de los clientes, otros datos relacionados con su puesto, como el departamento en el que trabajan o el correo corporativo del analista al cual se remitirán sus incidencias.

RF-2.– El sistema tendrá que solicitar al usuario la cuenta de correo electrónico al comienzo de la conversación.

RF-3.– Una vez el usuario proporciona el correo electrónico, el sistema comprobará que la dirección tenga un formato válido. En el supuesto de que el formato sea incorrecto informará al usuario y volverá a pedirle que escriba su dirección de correo. Si, por otro lado, el formato es válido el sistema debe asegurarse, mediante otro mensaje, de que el usuario ha escrito correctamente la cuenta de correo.

RF-4.– Cuando el usuario ha confirmado su dirección de correo, el sistema debe verificar que se encuentra almacenado en la base de datos. A continuación, el sistema realizará las siguientes acciones dependiendo del resultado de esta verificación:

RF-4.1.– Si el sistema no reconoce la cuenta de correo proporcionada informará al usuario y le permitirá introducirlo de nuevo.

RF-4.2.– En caso de que el usuario haya proporcionado una cuenta de correo que se encuentre almacenada y su rol no sea “analista”, el sistema comenzará la parte del diálogo en el que solicita la información necesaria acerca de la incidencia que el cliente quiere reportar.

RF-4.3.– Si la dirección de correo electrónico está almacenada en el sistema y pertenece a un usuario cuyo rol es “analista”, el sistema debe solicitar al usuario el identificador de la incidencia de cuya resolución pretende informar.

RF-5.– En caso de que el gestor o analista acaben de facilitar el identificador de una incidencia, el sistema debe comprobar que existe una incidencia no resuelta almacenada con dicho identificador. Tras esto, en caso de que encuentre una incidencia cuyo identificador corresponda con el proporcionado, el sistema debe comenzar la parte del diálogo destinada a recopilar la información del proceso de resolución.

RF-6.– Una vez se ha identificado correctamente un cliente, el sistema deber realizar preguntas que guíen al usuario de modo que proporcione la siguiente información:

- Fecha y hora en que se produjo la incidencia.
- Servicio o programa que el usuario estaba utilizando cuando ocurrió el problema.
- Características de la incidencia.

RF-7.– Cuando el cliente haya respondido a la última pregunta realizada por el sistema, éste debe asignar un identificador a la conversación, guardarla como incidencia no resuelta y generar un correo electrónico. Finalmente, el correo se enviará a la dirección de correo del analista asignado a este usuario.

RF-8.– El correo electrónico que se genere una vez que un cliente finaliza una conversación adecuadamente contendrá un fichero con la conversación completa, así como los datos del usuario. Se adjuntarán también los ficheros con las conversaciones con los analistas de las cinco incidencias solucionadas más relacionadas con la recién reportada. Por otro lado, el asunto del correo debe contener el identificador de la incidencia. En el cuerpo del mensaje aparecerán de nuevo los datos del usuario y el identificador de la incidencia, así como, los identificadores de cinco incidencias similares y enlaces que puedan resultar de ayuda.

RF-9.– Después de que el analista o gestor se hayan identificado con éxito y hayan proporcionado un identificador correcto, el sistema realizará preguntas que permitan al usuario informar sobre:

- El proceso que se llevó a cabo para solucionar la incidencia.
- El tiempo dedicado por el usuario para resolverla.

- Las posibles dificultades a las que se enfrentó el gestor o analista.

RF-10.– Una vez el analista responde a la última pregunta, el sistema relacionará la conversación con la incidencia que el analista considera resuelta y guardará ésta conversación como incidencia resuelta.

RF-11.– El sistema debe responder siempre, de modo que si no comprende la respuesta del usuario debe informarle y a continuación repetir la última pregunta.

RF-12.– El sistema debe ser coherente cuando varios usuarios tienen una conversación con él al mismo tiempo. El sistema debe ser capaz de seguir la lógica de las conversaciones cuando éstas se producen simultáneamente.

3.2. Requisitos no funcionales

RNF-1.– Portabilidad: El chat podrá incluirse en páginas web y se podrá acceder a él también desde una aplicación móvil.

RNF-2.– Mantenibilidad: Será modular de modo que se podrá cambiar la lógica conversacional sin editar el programa principal, así como el modo de comparación de las incidencias a la hora de recomendar otras similares.

RNF-3.– Usabilidad: Se implementará de modo que el chat sea intuitivo, utilizando lenguaje natural a la hora de recopilar la información.

RNF-4.– Seguridad: El sistema será seguro frente a inyección de código.

3.3. Arquitectura

El diagrama 3.1 de arquitectura pretende mostrar los componentes principales, así como la relación existente entre ellos.

Todos los datos que introducen los usuarios en el sistema se recogen mediante el canal de entrada. Este canal de entrada contará con una interfaz que permita recoger los mensajes del usuario y enviarlos al gestor de chat. Después de que el gestor de chat procese el mensaje devolverá la respuesta al canal de entrada que tendrá que ser mostrada a través de la interfaz previamente mencionada al usuario.

El gestor de chat tiene la responsabilidad de conectar todos los componentes de nuestro sistema. Como ya hemos descrito, se encarga de recibir y devolver mensajes al canal de entrada, pero también se encarga de enviar los datos de las incidencias procesados y posibles soluciones a éstas al canal de salida. El gestor de chat es el responsable de que todos los datos se guarden de forma correcta en la base de datos. Sin embargo, la función principal de este componente es la administración de las

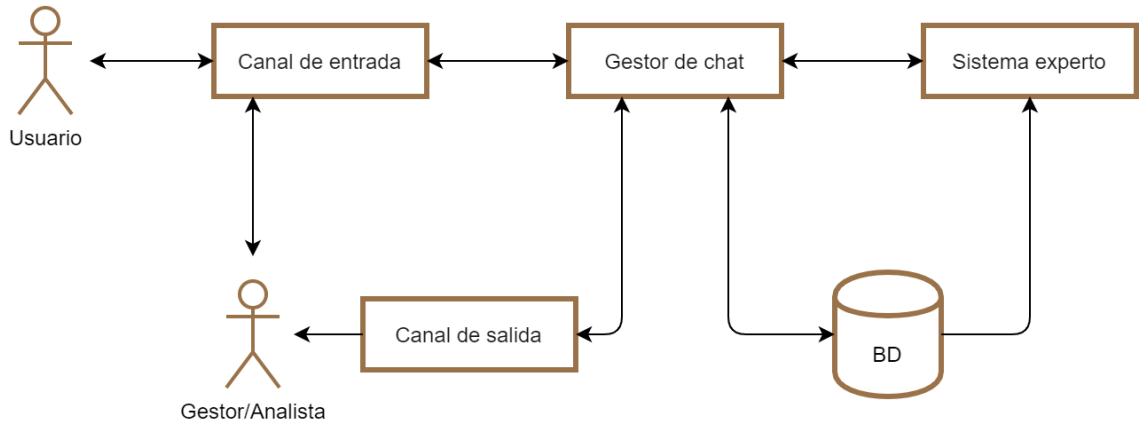

Figura 3.1: Arquitectura del sistema

conversaciones durante el desarrollo de estas. Es necesario también que el gestor de chat conozca el tipo de usuario al que pertenece la conversación, así como otros datos que puedan ser relevantes, para ello hace uso de la base de datos.

La base de datos contendrá toda la información necesaria para que el gestor de chat, cree mensajes correctos y procese los datos de los usuarios.

El sistema experto accederá a la base de datos para recomendar soluciones a la incidencia que se acaba de realizar basándose en las cinco incidencias más parecidas. También se encargará de mostrar, mediante sistemas de monitoreo externos, las alarmas que se detecten en el sistema en la franja horaria de la incidencia. El gestor de chat recibe estas recomendaciones y las envía al canal de salida.

Por último, el canal de salida se encargará de mostrar al analista los datos que el gestor de chat le haya especificado.

Como podemos observar, los usuarios tienen acceso al canal de entrada, pudiendo recibir y enviar mensajes a través de él. Sin embargo, solo el analista podrá hacer uso del canal de salida, y éste será limitado, ya que el analista tendrá acceso a los datos proporcionados por dicho canal, pero no podrá enviar datos mediante este componente, para ello deberá utilizar el canal de entrada.

Esta arquitectura permite aislar las interfaces (canales de entrada y salida) que utilizan los usuarios de la lógica de las conversaciones. De este modo, el canal de entrada recibe los mensajes del usuario y los envía, siempre con un mismo formato, al gestor de chat. Esto permite que el gestor de chat se encargue únicamente de mantener el estado de las conversaciones, procesar los mensajes de los usuarios, producir las respuestas y conectar los distintos componentes. Por otro lado, el sistema experto abstrae la parte inteligente del sistema de modo que pueda ser modificada sin afectar al resto de módulos.

4

Diseño y desarrollo

Ahora que hemos definido qué debe hacer el proyecto queda explicar cómo va a realizarse.

Comenzaremos mostrando qué tecnologías hemos empleado finalmente y los motivos que nos han llevado a tomar estas decisiones.

Por otra parte, se podrán observar las modificaciones finales de la arquitectura del proyecto y se describirá el diseño interno de cada componente.

4.1. Decisiones de diseño

Se pretende que los usuarios puedan acceder al sistema incluso cuando la red local de la empresa sufra algún incidente o incluso hayan perdido el acceso a internet en la oficina, por ello, hemos elegido que el acceso pueda realizarse a través de la aplicación de móvil de mensajería Telegram y la página web de la empresa. Consideramos que Telegram era una buena elección porque que está preparado para la creación de chatbots, lo que nos permitía conectarlo fácilmente con nuestro programa a través de Botpress.

Botpress es una herramienta muy flexible, lo que hace posible enlazarla con nuestro propio código. Esta herramienta no permitía, por ella sola, crear la lógica conversacional que cubriese todos los requisitos, pero si era útil para conectar el gestor de chat con las aplicaciones de *frontend*.

En este sistema, el gestor de chat actúa del lado servidor de nuestra aplicación, por lo que se ha utilizado como lenguaje Node.js. Por ello y por compatibilidad con otros sistemas de la empresa, se ha decidido emplear CouchDB como base de datos de nuestro proyecto. Esta base de datos, al ser NoSQL, nos aporta la flexibilidad necesaria para modificar la estructura de los datos cuando los cambios futuros del programa así lo requieran e incluso tener varios datos del mismo tipo almacenados con una estructura diferente.

Por otro lado, todo usuario analista o gestor debe tener una cuenta de correo corporativa, por ello se eligió un gestor de correo como canal de salida. Concretamente se ha utilizado el servicio Gmail para enviar los mensajes al analista ya que Node.js cuenta con una librería destinada a este fin.

Para realizar las recomendaciones que se enviarán al analista, al principio, nos planteamos utilizar Keras para el desarrollo de redes neuronales que analizasen los datos e intentasen predecir las soluciones de las incidencias. Sin embargo, no contábamos con datos con los que crear, entrenar y ajustar los parámetros de estas redes, por lo que finalmente no se empleó Keras. Por este motivo, tampoco se desarrolló código en Python.

La métrica tf-idf por otra parte, tiene en cuenta menos parámetros que las redes neuronales para la toma de decisiones, no obstante, no necesita de mucho volumen de datos para relacionar incidencias, por lo que puede ser una buena opción para las primeras fases de implementación del proyecto, mientras se obtiene el volumen de datos necesario.

4.2. Arquitectura

Como se ha explicado en el apartado anterior la arquitectura final, representada en la figura 4.1, se modificó incluyendo un nuevo módulo, Botpress.

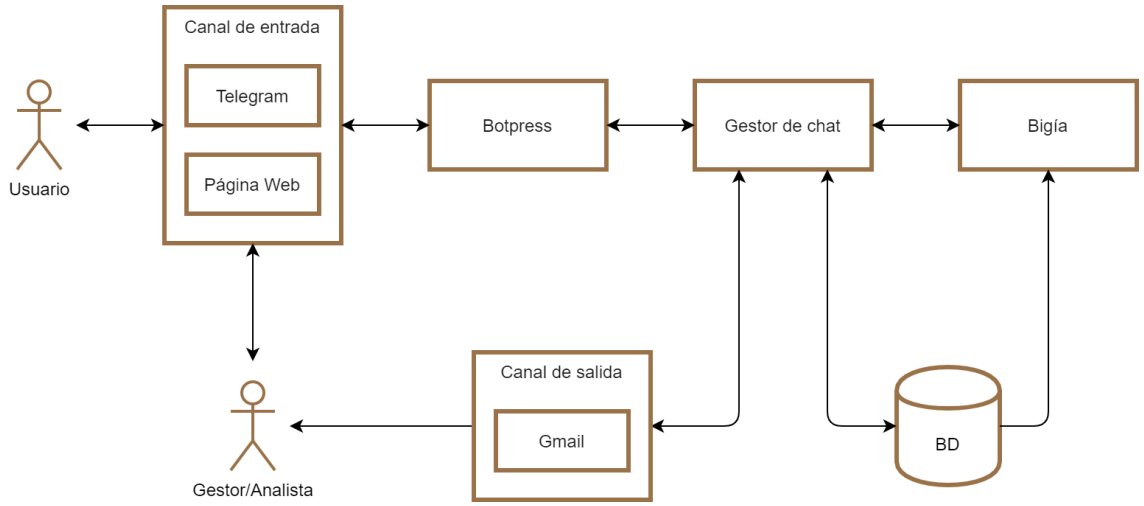

Figura 4.1: Arquitectura del sistema

Esta arquitectura presenta dos tipos de canales de entrada, esto permite realizar la conversación a través de la aplicación de móvil Telegram o mediante la página web corporativa.

Botpress se encarga de abstraer el canal de entrada permitiendo que el gestor de chat trabaje a un mayor nivel. Detecta cuando se ha enviado un mensaje y reconoce a que ventana de chat pertenece. Dicho mensaje se envía al gestor de chat, y una vez se recibe la respuesta, muestra correctamente el mensaje en la ventana de chat correspondiente.

Lo que antes era el sistema experto, ahora se ha renombrado como Bigía, que es el nombre del proyecto, debido a que en este componente se desarrolla la parte inteligente del proyecto. Este componente se encarga de recomendar distintas soluciones y de facilitar el trabajo del analista a la hora

de resolver la incidencia. Por un lado, Bigía utiliza los datos de las incidencias ya solucionadas para recolectar las más parecidas empleando la medida tf-idf nombrada en el capítulo 2.2 y detallada en la sección 4.3.3. Además, tiene la función de generar enlaces a otros sistemas de monitoreo de la empresa que muestran las alarmas detectadas en la franja horaria de la incidencia.

La base de datos CouchDB contiene los documentos necesarios para realizar todas las posibles conversaciones, así como los registros de las conversaciones que ya se han realizado con los usuarios. También se encarga de guardar los datos de todos los usuarios clientes, gestores y analistas.

Por último, como ya se ha nombrado anteriormente, para comunicar los datos al analista el gestor de chat hace uso del gestor de correo Gmail.

4.3. Diseño interno

Hasta ahora hemos visto las funcionalidades que debe cumplir el proyecto y la arquitectura, con cada uno de sus componentes, los cuales se encargarán de suplir dichas funcionalidades.

En este apartado vamos a explicar más detalladamente cómo se va a realizar el código que realmente llevará a cabo las acciones necesarias para que los requisitos especificados se cumplan, así como asegurar el buen funcionamiento del proyecto.

4.3.1. Canales de entrada y Botpress

Los canales de entrada, Telegram y la página web de la empresa, no han sido desarrollados en este proyecto, si no que se han utilizado para llevar a cabo el nuestro.

Gracias a la plataforma Botpress podemos crear fácilmente una conexión entre el gestor de chat y estos canales de entrada. En el caso de Telegram, hemos creado un bot (aféresis de robot) mediante la cuenta *@BotFather* de Telegram, la cual genera y nos proporciona un identificador necesario para que Telegram y Botpress se relacionen.

Por otra parte, para que la interfaz de chat se encuentre en la página web de la empresa debemos, primero, instalar el módulo *@botpress/channel-web* en Botpress y, a continuación, insertar el código JavaScript 4.1 en la página web.

Código 4.1: A continuación se muestra el código JavaScript genérico necesario para incrustar la interfaz web de Botpress en una página web.

```
<script src="<host>/api/botpress-platform-webchat/inject.js"></script>
<script>window.botpressWebChat.init({ host: '<host>' })</script>
```

Botpress consta de una página de desarrollador donde poder realizar ciertos cambios. Botpress te permite configurar la página de tal modo que tengas que identificarte si intentas acceder a la interfaz de desarrollador, por lo que se necesita una cuenta y contraseña de desarrollador.

Por otro lado, para que la plataforma Botpress pueda enviar los datos al servidor se deben configurar algunos parámetros dentro del código de Botpress. Para que el gestor de chat y el canal de entrada se comuniquen correctamente se tiene que desarrollar una función en Botpres que, por cada mensaje recibido del canal de entrada Botpress lo reenvíe al gestor de chat y, una vez el gestor haya enviado la respuesta a Botpress, se muestre en la interfaz del chat.

4.3.2. Gestor de chat

Como se ha explicado anteriormente, el gestor de chat está escrito en Node.js, basado en el lenguaje orientado a objetos y asíncrono JavaScript, para aplicaciones *backend*. Por ese motivo, podemos desarrollar el código de forma más organizada mediante la creación de clases. A continuación, en la figura 4.2, podemos ver las distintas clases elegidas para gestionar las distintas conversaciones que puedan producirse, es decir, aquellas que contendrá el gestor de chat.

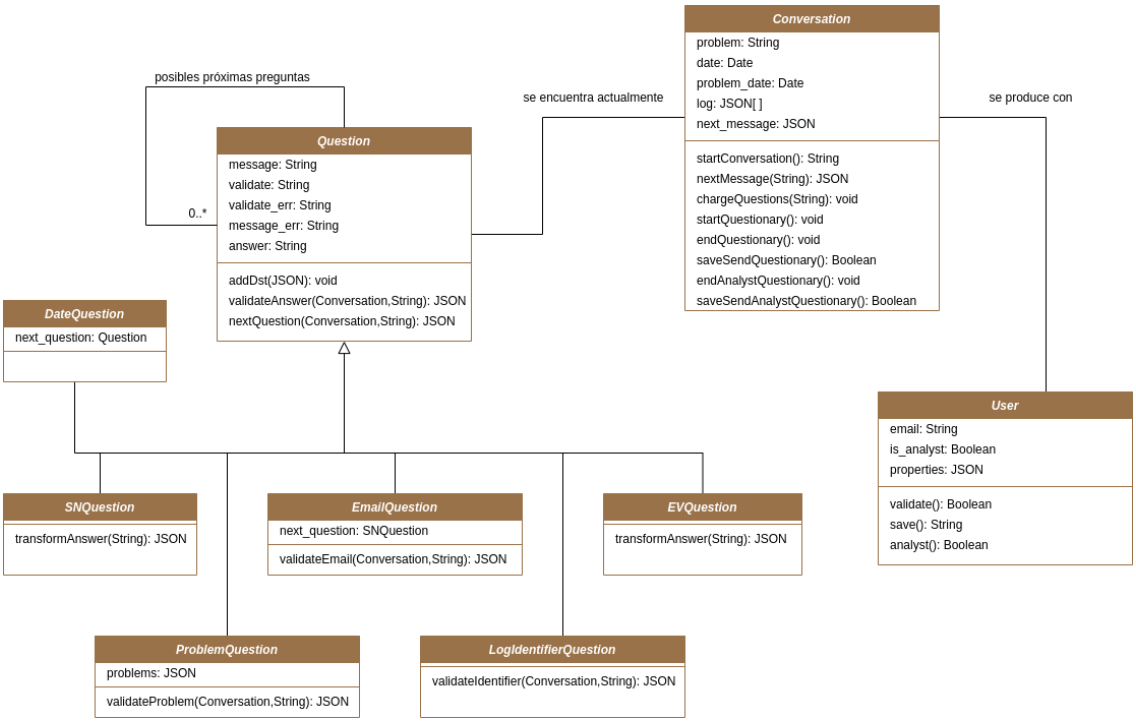

Figura 4.2: Representación de las clases que conforman el gestor de chat

Como podemos ver en la figura 4.2, las conversaciones tienen una sola pregunta, esto se debe a que la única pregunta que contiene es la actual, la que va a realizarse justo en ese momento o la que

se acaba de realizar, dependiendo del estado en que se encuentre la conversación, es decir, de si el usuario ha respondido o no a dicha pregunta. El resto de preguntas se encuentran enlazadas a otras formando un árbol, por tanto, cada pregunta contiene las siguientes posibles preguntas junto con las condiciones que les harán llegar a estas. Para comprender mejor esta forma de avanzar a través de la conversación es importante explicar la estructura de las conversaciones, esto lo veremos en la sección 4.3.4.

Existen siete tipos de preguntas dependiendo de las respuestas que permiten y de las acciones que deben realizar en base a ellas:

- *Question*. Este tipo de pregunta es genérica, admite cualquier texto y no realiza una validación concreta de la respuesta, si no que esta validación se define al crear el objeto.
- *DateQuestion*. Esta pregunta está diseñada para obtener una fecha y una hora. Para poder preguntar por la hora necesitamos que esta pregunta contenga una segunda. La pregunta original se enarga de solicitar la fecha, para ello valida una primera respuesta que debe cumplir un formato específico de fecha (Ejemplo: “10/5/2020”). La segunda pregunta, de tipo *Question*, se encarga de validar la hora.
- *SNQuestion*. Ésta es una pregunta binaria, que espera como resultado “si” o “no”.
- *EmailQuestion* Como respuesta a esta pregunta esperamos una dirección de correo electrónico. Esta clase validará que el formarto de la dirección de correo sea correcto y en ese caso pasará a la pregunta de tipo *SNQuestion* que contiene. Esta segunda pregunta tiene como función asegurarse de que el usuario ha escrito la dirección de correo electrónico deseada.
- *EVQuestion* Esta clase acepta como respuesta un número del 1 al 5. Su uso está pensado para preguntas relacionadas con la satisfacción de un usuario al utilizar cierto programa o plataforma.
- *ProblemQuestion*. En este caso, esta clase tiene un uso bastante concreto, ya que que permite una respuesta entre múltiples opciones que están predefinidas. Se encarga de validar que la respuesta sea una de las opciones mostradas al usuario o el número que preceda a alguna de las opciones.
- *LogIdentifierQuestion*. Esta pregunta obtiene como respuesta un identificador de incidencia y, por supuesto, debe validar que este sea correcto.

Por otro lado, una vez la conversación con un cliente finaliza el gestor de chat hace uso del módulo Bigía para obtener las incidencias recomendadas y los enlaces a los sistemas externos de monitoreo con los que, junto a otros datos que posee el gestor de chat, genera un correo que será enviado al analista a través de Gmail, mediante la librería *nodemailer*.

4.3.3. Bigía

El módulo Bigía, como ya se ha explicado, comprenderá la parte inteligente del proyecto. En este componente se busca recomendar al analista soluciones a incidencias a través de distintas herramientas. La que se van a emplear en esta primera iteración del proyecto son la métrica tf-idf para recomendar incidencias similares y la generación de enlaces a otros sistemas de monitoreo.

Bigía se encargará de recomendar las incidencias similares. Existe una multitud de métricas tf-idf que podrían emplearse para comparar documentos, nosotros hemos utilizado la descrita en la ecuación

4.1.

$$\begin{cases} \mathrm{tf}(t,d) = 1 + \log_2(\mathsf{frec}(t,d)) \\ \mathrm{idf}(t) = \log_2\left(\frac{D}{D_t}\right) \end{cases} \tag{4.1}$$

A continuación se definen las distintas variables de la ecuación 4.1:

- d es el documento que estamos analizando.
- t se refiere a una palabra (o término) del documento.
- frec(t, d) es la frecuencia de t en d, es decir, el número de veces que aparece el término t en el documento d.
- D es el número total de documentos de que disponemos.
- D_t es el número de documentos que contienen el término t.

Para comparar los distintos documentos entre sí se ha empleado un modelo vectorial. Por este motivo, creamos un vector por cada documento donde cada coordenada del vector será la medida tf-idf de cada palabra para ese documento, como puede verse representado en la ecuación 4.2, donde el conjunto total de palabras o vocabulario es $T = \{t_1, t_2, ..., t_n\}$.

$$q_d = (\mathrm{tf}(t_1,d) \cdot \mathrm{idf}(t_1), \mathrm{tf}(t_2,d) \cdot \mathrm{idf}(t_2), ..., \mathrm{tf}(t_n,d) \cdot \mathrm{idf}(t_n)) \tag{4.2}$$

Una vez se han calculado los vectores, uno para cada uno de los documentos (d_1 y d_2), podemos compararlos mediante la ecuación 4.3, de modo que cuanto mayor es este coseno, más se asemejan los vectores.

$$\cos(q_{d_1}, q_{d_2}) = \frac{q_{d_1} \cdot q_{d_2}}{|q_{d_1}| \cdot |q_{d_2}|} \in [0,1] \tag{4.3}$$

Si realizamos esta comparación con todas las incidencias resueltas y la que acaba de realizarse podemos comprobar cuál de las previamente resueltas es más parecida a ésta última. Se recogerán las cinco incidencias previas que más relación tienen con la actual y se enviarán al gestor de chat pata que pueda mandarlas al analista.

Bigía utiliza la última respuesta del cuestionario para realizar esta comparación, dado que es el momento en el que se pide al usuario que detalle la incidencia mediante texto libre, sin opciones predefinidas.

Para evitar que se tengan en cuenta palabras como artículos, conectores o incluso símbolos, se realiza una lista con ellas para que no se descarten antes de realizar las comparaciones. Se ha empleado una librería de Node.js llamada *tokenize-this* que separa el texto libre en palabras para, después, eliminar aquellas que no deben ser tenidas en cuenta para el cálculo tf-idf.

Además de recomendar las incidencias similares, este módulo se encarga de generar un enlace a los sistemas de monitoreo de la empresa. En dicho enlace se podrán conocer las alarmas que se han detectado en el sistema en un tiempo cercano a la incidencia reportada, de modo que el analista pueda comprobar rápidamente si alguna de estas alarmas tiene relación con la incidencia.

4.3.4. Base de datos

Por los motivos ya explicados en la sección 4.1, el tipo de base de datos NoSQL elegida ha sido CouchDB y, por ello, las conversaciones se crean mediante JSONs que contienen un array de preguntas, las cuales se almacenan en documentos en la base de datos. Las preguntas están compuestas por un identificador, el texto de la pregunta, el tipo de ésta, el mensaje de error que se mostrará en caso de que la respuesta no sea válida y las distintas preguntas posibles que se harán tras esta, junto con las condiciones que deben cumplirse para pasar a cada una de las siguientes preguntas. Para permitir introducir cualquier tipo de condición ésta se define mediante una cadena de caracteres, que representará un fragmento de código con respuesta Booleana, que será evaluada finalmente por el gestor de chat.

La base de datos también contiene los datos de todos los usuarios clientes, analistas y gestores introducidos mediante JSON's. El único dato que el gestor de chat exigirá de estos usuarios es que contengan una dirección de correo electrónico. Los clientes se almacenan separados de los analistas y gestores, de modo que no se necesiten más datos, bastará con que el gestor de compruebe si la dirección de correo electrónico proporcionada se encuentra en la base de datos de analistas o de clientes.

Como ya se ha explicado, todos los datos de CouchDB se almacenan en documentos mediante JSONs, por consiguiente, las conversaciones de incidencias sin resolver y resueltas también se guardan de esta forma en CouchDB. Sin embargo, todas estas conversaciones se almacenan también como ficheros fuera de la base de datos CouchDB para enviar estos ficheros, cuando se considere oportuno, por correo electrónico al analista o gestor.

5

Pruebas y resultados

5.1. Pruebas

Como se ha explicado en el análisis, este proyecto sigue un modelo de desarrollo iterativo, pero cada etapa es lineal. Por este motivo, una vez que el sistema está desarrollado procedemos a probarlo.

Debido a que el sistema es completo y cuenta con ciertos módulos sencillos realizaremos únicamente pruebas de caja negra.

Lo primero que debemos hacer es iniciar el gestor de chat, guardar los datos en la base de datos, iniciar botpress y crear las conexiones. Es decir, iniciamos los contenedores que hemos creado mediante Docker.

Empezamos probando que los canales de salida se muestran correctamente y que se han conectado al gestor de chat. Para ello accedemos a la página web y a la aplicación móvil Telegram y procedemos a comenzar una conversación.

Después, realizamos distintos hilos de conversación con un usuario tipo cliente para asegurarnos de que el sistema es capaz de seguir la lógica de la conversación, de modo que sea coherente.

A continuación, ingresaremos en un gestor de correo electrónico con la dirección de correo del analista correspondiente a dicho usuario con la intención de asegurarnos de que el mensaje ha llegado y contiene la información correcta.

Si repetimos este proceso varias veces podemos comprobar si el sistema es capaz de proporcionar las recomendaciones al analista del modo en que esperábamos. Para realizar este proceso se resolverán 14 incidencias con textos libres que permitan ver las ventajas y desventajas de la métrica tf-idf. Tras resolverlas, se repotará una última incidencia y se mostrarán las cinco que el sistema ha considerado más parecidas.

Gracias al entorno en que se desarrolló este proyecto, pudimos realizar pruebas con distintos usuarios de la empresa, de los que obtuvimos una retroalimentación. Ésta nos sirvió para realizar ciertas mejoras que nombraremos más adelante.

5.2. Resultados

La interfaz ha quedado como esperábamos, incluso hemos podido añadir el logotipo y los colores de la empresa como se observa en las figuras 5.1, 5.2 y 5.3.

Figura 5.1: En esta se muestra la interfaz web con el chat sin desplegar

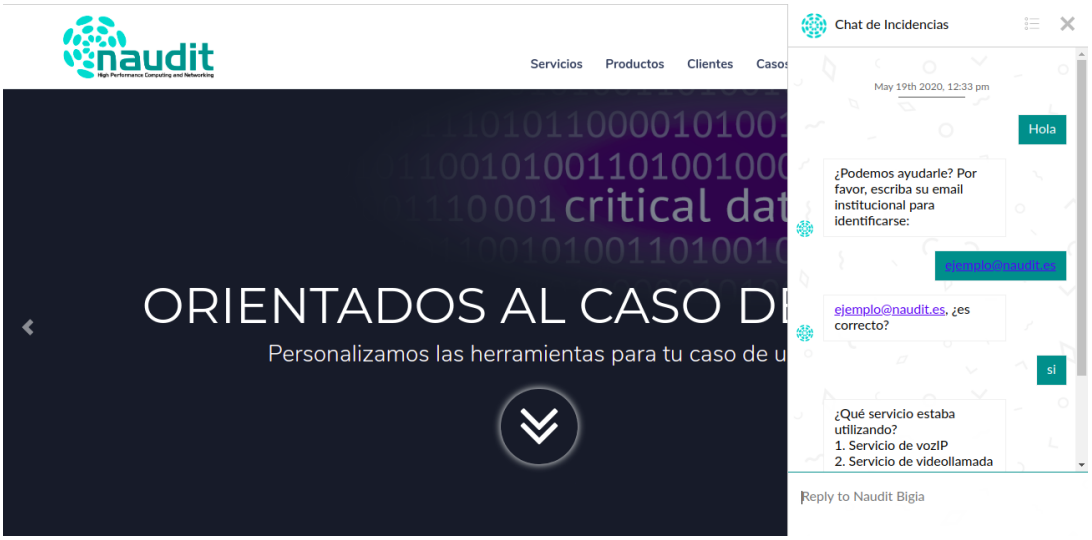

Figura 5.2: En esta imagen podemos ver un ejemplo de conversación realizado en la interfaz web de la empresa

Además, en la figuras 5.2 y 5.3, podemos ver cómo el gestor de chat responde a nuestros mensajes, por lo que concluimos que la conexión a través de Botpress se ha creado correctamente.

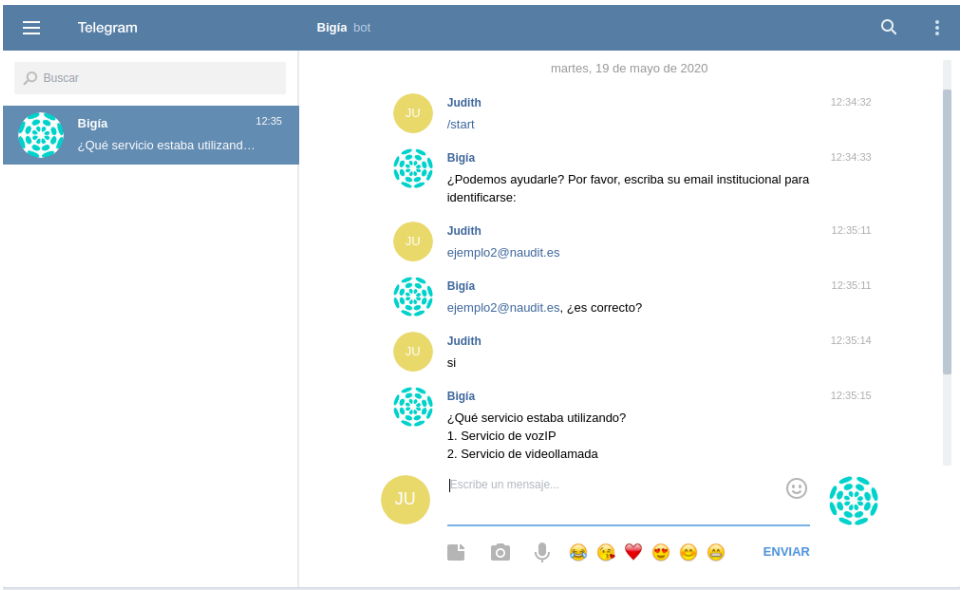

Figura 5.3: En esta figura se muestra un ejemplo de conversación realizado en la aplicación de Telegram

A continuación, mostraremos dos hilos de conversación con los que se muestra que el programa es coherente y capaz de seguir la lógica de la conversación.

La primera conversación mostrada en la figura 5.4 informa de un problema con la sincronización del audio y el vídeo durante una videollamada. La figura 5.5 muestra la segunda conversación, en la cual un cliente intenta realizar una llamada de *VoIP* pero no tiene conexión a internet.

Figura 5.4: Primer ejemplo de conversación con el sistema. Para que sea más claro se ha eliminado la parte de la interfaz web menos relevante.

Además, se ha comprobado que el analista recibe el correo electrónico con los datos necesarios. Esto se muestra en la imagen 5.6, en la que se puede observar que el formato era el esperado, los datos más importantes se encuentran en el cuerpo del mensaje y el fichero de la conversación está adjunto. Esta conversación se realizó antes de solucionar ninguna incidencia, por lo que vemos que en este caso no hay incidencias recomendadas.

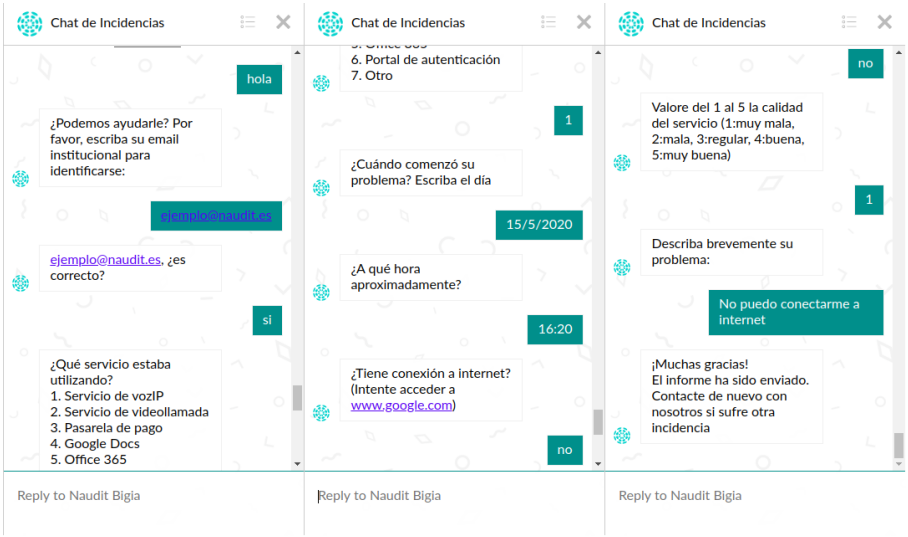

Figura 5.5: Segundo ejemplo de conversación con el sistema. Al igual que en la anterior conversación, mostramos solo la parte del chat de la interfaz.

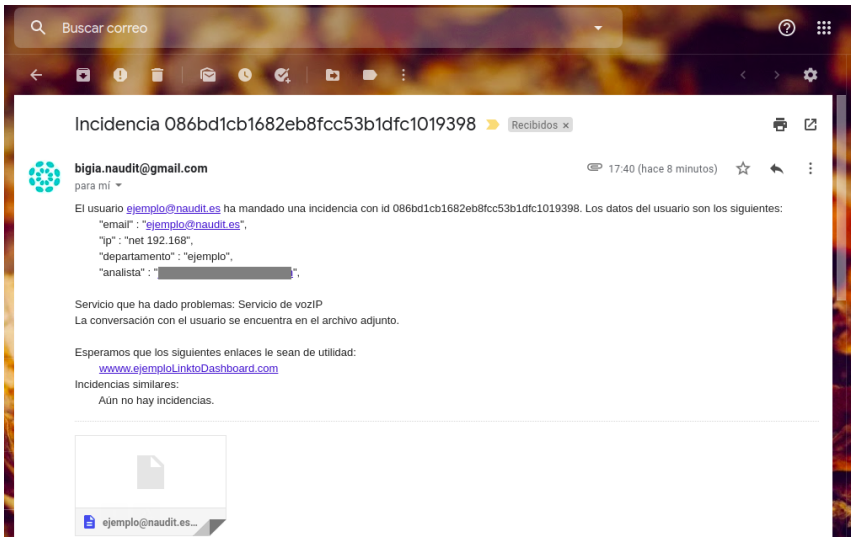

Figura 5.6: En esta imagen se muestra un gestor de chat (Gmail) en el que se ha recibido el mensaje de la incidencia 5.5.

El sistema es capaz de responder correctamente a preguntas con un formato incorrecto o direcciones de correo que no se encuentren en el sistema, y no permite en este segundo caso, que el usuario avance en la conversación. Estas acciones pueden observarse en las figuras 5.7 y 5.8, respectivamente.

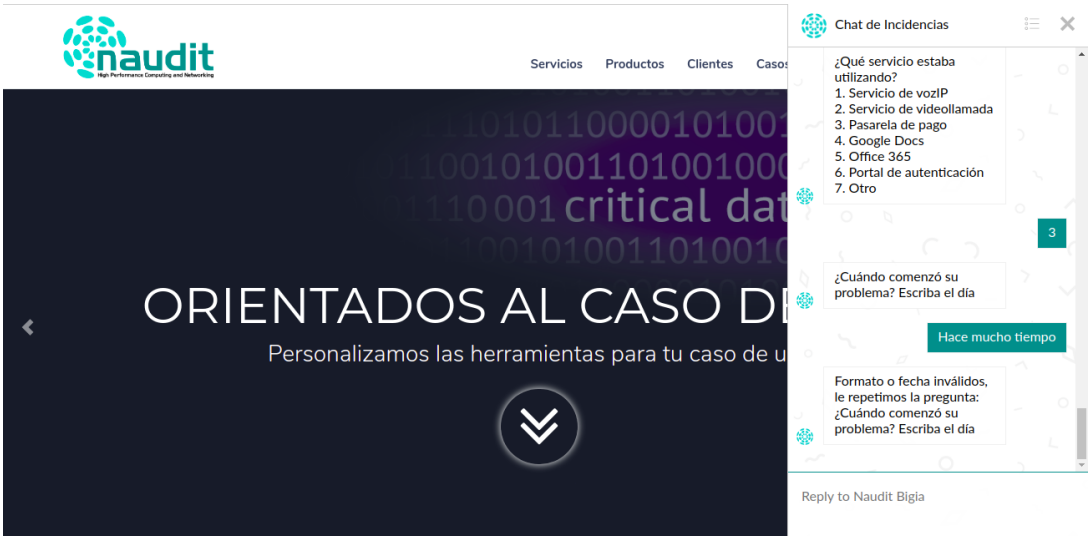

Figura 5.7: Esta figura muestra cómo el gestor de chat informa al usuario cuando el mensaje no coincide con el formato esperado para una fecha.

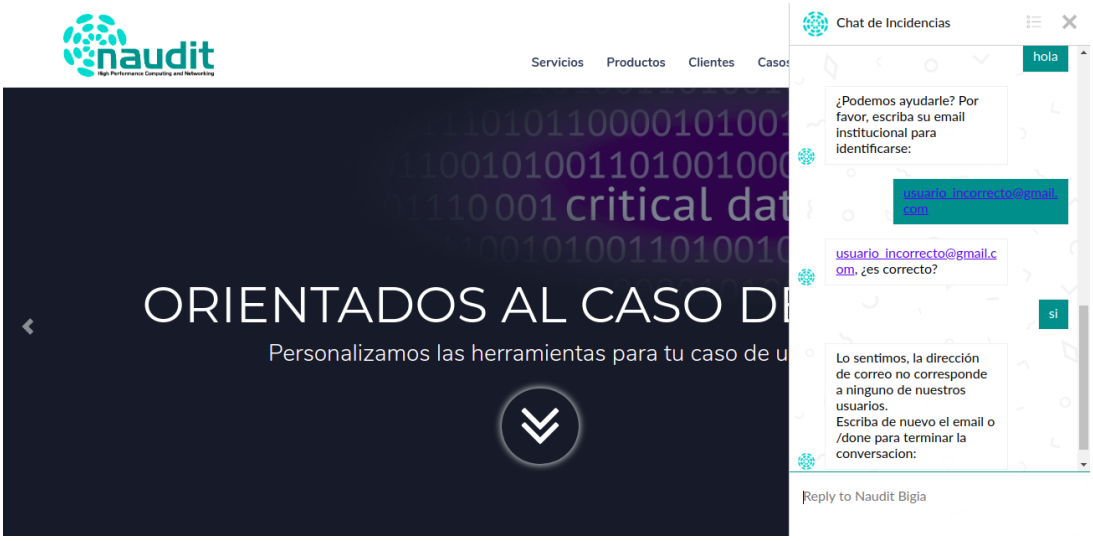

Figura 5.8: Podemos ver como el sistema no permite que un usuario no almacenado en la base de datos acceda a la conversación.

A continuación, hemos realizado 14 conversaciones con el sistema. En la tabla 5.1 se muestra el nombre del archivo que produce cada incidencia y el texto libre escrito por el analista. El resto de datos de la conversación no se incluyen dado que Bigía solo tiene en cuenta el texto libre. Estas conversaciones se han comparado con una última cuyo texto libre es “Durante la videollamada la imagen funcionaba pero el sonido no”.

Orden	Nombre del archivo	Texto libre
1	ejemplo@naudit.es_1589910553925	Durante la videollamada la imagen funcionaba pero el sonido no
2	ejemplo@naudit.es_1589910727856	Durante la videollamada el sonido funcionaba pero la imagen no
3	ejemplo@naudit.es_1589910779944	Durante la videollamada la imagen y el sonido fallaban
4	ejemplo@naudit.es_1589911131934	El documento no se guarda
5	ejemplo@naudit.es_1589911218734	No tengo internet
6	ejemplo@naudit.es_1589911302889	La llamada se ha cortado
7	ejemplo@naudit.es_1589911399649	No me deja autenticarme
8	ejemplo@naudit.es_1589911487770	La imagen tiene mala calidad
9	ejemplo@naudit.es_1589911566526	El sonido llega con retardo
10	ejemplo@naudit.es_1589911679540	No puedo editar mis archivos de google drive
11	ejemplo@naudit.es_1589911831246	Se me ha cortado la conexión
12	ejemplo@naudit.es_1589911906600	No escucho a la persona a la que he llamado
13	ejemplo@naudit.es_1589911995250	No he podido comenzar la videollamada
14	ejemplo@naudit.es_1589912247073	El pago no se ha realizado correctamente

Tabla 5.1: Esta tabla nos permite ver los datos de las 14 conversaciones con las que compararemos la nueva incidencia. La primera columna muestra el orden en que se realizaron estas incidencias, la segunda el nombre del archivo que se genera tras la conversación y la última el texto libre de la conversación.

Tras realizar la conversación de la incidencia que queremos comparar, accedemos al gestor de correos del analista, figura 5.9, y comprobamos que, por este orden, las incidencias más parecidas son 1,2,3,8,9. Todas estas incidencias comparten palabras con la que estamos comparando, es decir, tienen cierta relación con la incidencia recién reportada. Esto era lo que esperábamos de esta métrica.

Sin embargo, hemos pensado en estas 14 incidencias con la intención de mostrar también las desventajas de utilizar tf-idf. Si nos fijamos, la incidencia número 2 tiene las mismas palabras que la que comparamos, pero el significado es el opuesto. Esto ocurre, como ya habíamos predicho, por que este tipo de comparaciones no tienen en cuenta el orden de las palabras ni el significado de estas.

Por otro lado, comprobamos también que el sistema funcionaba de forma coherente cuando se realizaban dos conversaciones distintas al mismo tiempo.

Durante las pruebas que pudimos realizar con usuarios recibimos comentarios bastante positivos, en general gustó la aplicación. Sin embargo, nos recomendaron algunas mejoras que finalmente se incluyeron:

- Se aceptan como formatos para fechas las palabras “hoy” y “ayer”.
- Se aceptan como formatos para las horas “H”, es decir, “8” sería un formato válido.

Gracias a la modularidad del sistema, este cambio solo afectó a dos funciones auxiliares que verificaban que las fechas y las horas tuvieran el formato correcto.

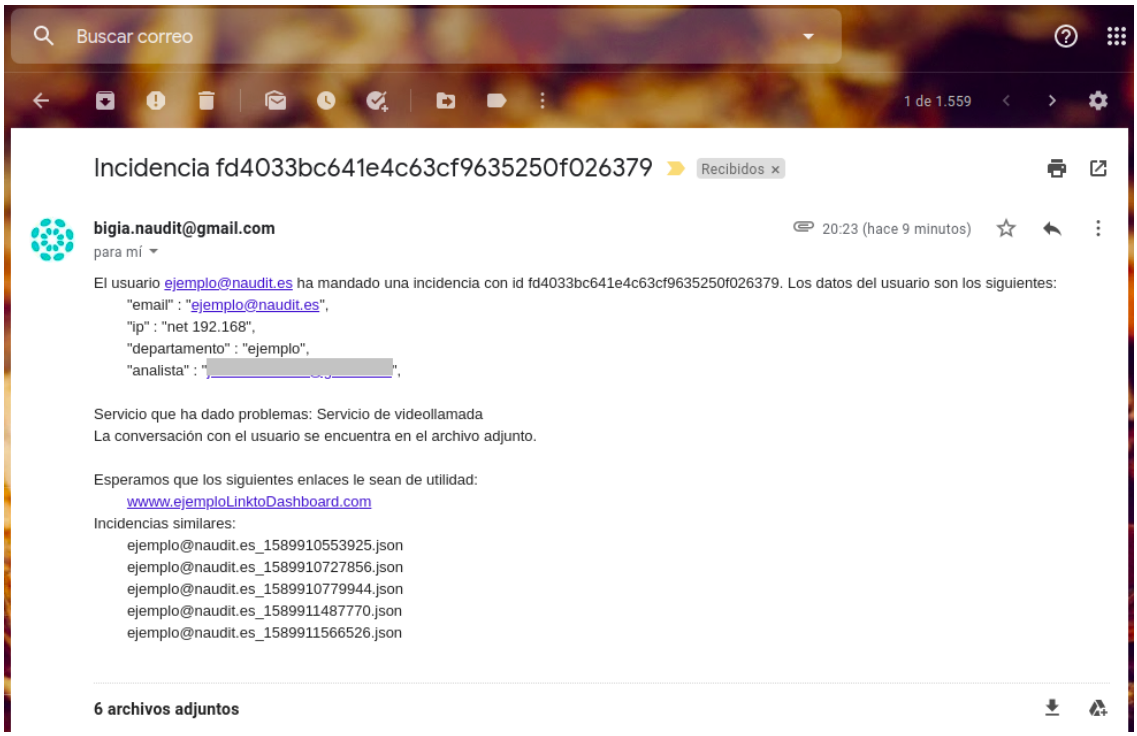

Figura 5.9: En esta figura se muestra un mensaje al analista con las cinco incidencias más parecidas de la tabla 5.1 al compararlas a una con texto libre: “Durante la videollamada la imagen funcionaba pero el sonido no”.

6

Conclusiones

6.1. Conclusiones y lecciones aprendidas

En este trabajo, se ha desarrollado con éxito el primer hito del proyecto de reporte y gestión de incidencias. Mediante el uso de diversas herramientas y servicios previamente mencionados hemos conseguido obtener un sistema básico y funcional que cubre todos los requisitos iniciales del cliente. Además, gracias a la arquitectura elegida y a la modularidad del sistema, se podrán introducir mejoras y nuevas funcionalidades con facilidad, favoreciendo la mantenibilidad del software realizado.

Este sistema se ha centrado en la usabilidad por parte de los clientes que deben reportar las incidencias, así como facilitar la labor del analista o del gestor de red. Las diversas pruebas y despliegues realizados ante usuarios nos han confirmado que se ha obtenido el grado de usabilidad deseado y que la información que obtienen los analistas es la requerida.

Gracias a este proyecto, he tenido la oportunidad de aprender un nuevo lenguaje, los principios de programación de un *backend*, bases de datos NoSQL que no había empleado antes y herramientas útiles para el desarrollo y despliegue de aplicaciones con una arquitectura de microservicios. Estas herramientas incluyen el procesamiento del lenguaje natural, métricas para medir la relevancia de documentos respecto a una consulta, librerías de aprendizaje profundo y servicios que permiten desplegar proyectos mediante contenedores. Esto me ha permitido ver el proceso completo de, desde 0, proponer y diseñar una arquitectura así como desarrollarla e incluso aprender a desplegarla.

Además de conocer nuevos lenguajes, servicios y herramientas informáticas, he podido afianzar muchos de los conocimientos que se muestran durante el grado, como lo son la importancia del análisis y diseño del *software*, así como múltiples técnicas empleadas durante los proyectos realizados durante las clases prácticas de las distintas asignaturas.

6.2. Contribuciones

Este trabajo se ha desarrollado en el ámbito empresarial gracias al convenio existente entre la Universidad Autónoma de Madrid y Naudit HPCN. Concretamente, este proyecto se ha ha llevado a cabo en el departamento Cátedra UAM-Naudit permitiendo recopilar opiniones de verdaderos analistas y datos de monitorización de infraestructuras industriales.

En naudit, he podido diseñar y desarrollar este proyecto por completo y desde el principio con la supervisión de analistas e ingenieros especializados en resolver incidencias de red. Además, este software fue desplegado, como prueba de concepto, en dos centros de procesamiento de datos. Se comenzó desplegando un prototipo del sistema en una empresa internacional del sector TIC y, a continuación, en otra del sector energético. La primera impresión fue muy buena, por lo que se espera poder llegar a implantar este servicio como apoyo al resto de sistemas de monitoreo existentes en dichas empresas para la detección de incidencias.

6.3. Trabajo futuro

Este trabajo desarrolla únicamente el primer hito del proyecto, es decir, crea una primera versión funcional del sistema deseado con un diseño de arquitectura que tiene como propósito permitir la implantación de nuevas funcionalidades y mejoras.

Existen una gran cantidad de posibilidades a la hora de añadir mejoras y funcionalidades a este proyecto en un trabajo futuro. Como ya hemos explicado en el trabajo, la herramienta Keras [11], para el aprendizaje profundo, permitiría realizar mejores predicciones una vez se han recopilado suficientes datos de incidencias resueltas. A corto plazo, se podría implementar un modelo vectorial para comparar todas las respuestas del cliente, no sólo las respuestas con texto libre, e incluso otros datos del usuario como su dirección IP o el departamento en el que trabaja.

Además, existe la posibilidad de implementar cierta inteligencia que guíe el orden de las preguntas realizadas basándose en el usuario que realiza la incidencia, incidencias producidas en un tiempo cercano o una combinación de estas alternativas. Para esto, al igual que para implementar una inteligencia profunda a través de redes neuronales, se precisa una recopilación de datos previa con la que entrenar estos modelos.

Este proyecto también puede modificarse con bastante facilidad con el objetivo de ser empleado en otros ámbitos empresariales, como podría ser el *marketing*. El sistema podría realizar preguntas con el propósito de recomendar ciertos productos o servicios de una empresa.

Bibliografía

[1] Instituto Nacional de Estadística, "Uso de TIC y comercio electrónico en las empresas. Año 2018-Primer trimestre de 2019." Online: https://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736176743&menu=ultiDatos&idp=1254735576692, 2019.

[2] Amazon Web Services, Inc, "Amazon CloudWatch: Observación de sus recursos y aplicaciones de AWS tanto en AWS como locales." Online: https://aws.amazon.com/es/cloudwatch/, 2020.

[3] Atlassian, "Transforma la TI para entregas más rápidas y una colaboración sin esfuerzo." Online: https://www.atlassian.com/es/software/jira/service-desk, 2020.

[4] BigPanda, Inc, "Event Correlation, powered by AIOps." Online: https://www.bigpanda.io/, 2020.

[5] Atlassian, "Modern incident management for operating always-on services." Online: https://www.atlassian.com/software/opsgenie, 2020.

[6] Deniz Mehmed, "Cuando utilizar NodeJS." Online: https://codigodiario.me/cuando-utilizar-nodejs/, 2016.

[7] M. Brown, *Getting Started with CouchDB*. O'Reilly Media, Inc, 2012.

[8] A. Sarasa, *Introducción a las bases de datos NoSQL usando MongoDB*. S.L. EDITORIAL UOC, 2016.

[9] Docker Inc, "What is a Container?." Online: https://www.docker.com/resources/what-container, 2020.

[10] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc, 2nd ed., 2019.

[11] Keras Team, "Keras: Simple. Flexible. Powerful." Online: https://keras.io/.

[12] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of Massive Datasets*. USA: Cambridge University Press, 2nd ed., 2014.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID